HoloLogger: Keystroke Inference on Mixed Reality Head Mounted Displays

Shiqing Luo[*]
George Mason University

Xinyu Hu[†]
Automatic Data Processing, Inc.

Zhisheng Yan[‡]
George Mason University

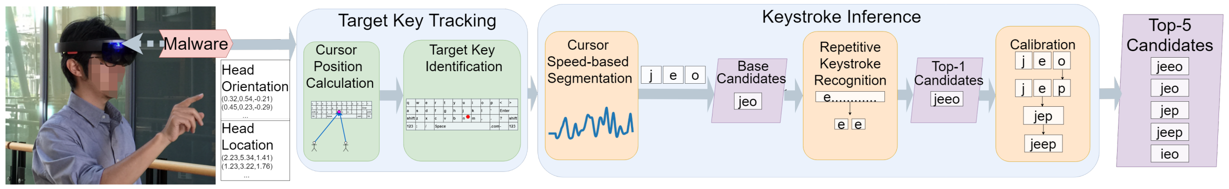

Figure 1: HoloLogger attack system. While a user is entering sensitive input in a mixed reality (MR) head-mounted display (HMD), malware in the benign App collects a data stream of head orientation and location. Then the proposed target key tracking scheme enables six-degree-of-freedom (6DoF) HMD tracking and generates a stream of keys aimed at by the user. Finally, based on the timing patterns of head motion and air taps in MR, the keystroke moments are identified, the keystrokes, including repetitive ones, are recognized, and the candidates of user input are inferred.

Abstract

When using personal computing services in mixed reality (MR) such as online payment and social media, sensitive information and account passwords must be typed in MR. To design secure MR systems and build up user trust, it is imperative to first understand the security threat to the sensitive MR input. Although keystroke inference attacks by analyzing human-computer interaction in videos or via wireless signals have been successful, they require placing extra hardware near the user which is easily noticeable in practice. In this paper, we expose a more dangerous malware-based attack through the vulnerability that no permission is required for accessing MR motion data. We aim to monitor MR headset motion and infer the user input through a benign App. Realizing the attack system requires addressing unique challenges in MR such as six-degree-of-freedom (6DoF) device motion and no explicit motion signal for keystroke identification. To this end, we present HoloLogger, the first malware-based keystroke inference attack system on HoloLens. HoloLogger is empowered by a 6DoF-head-motion-driven key tracking scheme and an air-tap-pattern-based keystroke inference framework. Extensive evaluations with 25 users and 750 inference trials of passwords consisting of 4–8 lowercase English letters demonstrate that HoloLogger successfully achieves a top-5 accuracy of 93%. HoloLogger is also robust in various environments such as different user positions and input categories.

Index Terms: Human-centered computing—Human computer interaction —Mixed/augmented reality; Security and privacy—Human and societal aspects of security and privacy—Privacy protection;

1 Introduction

Mixed Reality (MR) merges virtual objects with the real world, creating a new environment where computer-generated digital objects co-exist and interact with physical objects in real-time [26]. MR has the potential of transforming numerous applications from manufacturing and healthcare to education and entertainment [21]. As the market grows, a wide spectrum of personal computing services have been built into MR platforms such as online payment, email, and social media. MR users need to enter sensitive information such as credit card numbers, account passwords, account security questions while using these services [16]. If such personal information is leaked, there would be serious security and privacy issues. As an emerging technology, the security guarantee of MR is still in its infancy. In order to design secure MR systems and ultimately make MR a trustworthy technology for every consumer, we must first understand the potential vulnerability of sensitive MR data.

Keystroke inference attack that infers user input by analyzing human-computer interaction is one of such potential security threats to MR. In a typical MR system, users utilize novel interaction modalities such as head movement, hand movement, and gesture to navigate through a virtual keyboard and commit a keystroke. Since the entire input process is fully exposed to the public, attackers may capture and exploit the leaked information of user motion and device motion to infer the user input. This type of attack has been conducted successfully in computing platforms including MR [18], smartphones [10, 19], and virtual reality (VR) [1, 2, 6, 20] by capturing user/device motion through different side channels such as video recordings and wireless signals. However, these keystroke inference attacks require additional hardware such as cameras and wireless transceivers to record the user's motion. The extra hardware must be installed near the users without alerting them, which limits the potential attack scenarios. Therefore, the risks of hardware-assisted keystroke inference are reduced in practice.

In this paper, we expose a more dangerous malware-based keystroke inference attack in MR, where motion information will be analyzed to infer user input without requiring extra hardware. We observe that HoloLens system does not require user permission for an App to access its motion data. This vulnerability allows us to employ malware on a benign App to record the headset motion for keystroke inference. While this paper focuses on HoloLens with the head gaze and commit (HGC) input model, the principles and algorithms of the attack are generic. By obtaining the keyboard layout of a target device or App and updating the keyboard parameters in our framework, one can extend the attack to other MR devices supporting HGC. More importantly, anyone with basic knowledge of installing software and deploying an App can launch the attack. The ultimate goal of this paper is to call for immediate countermeasure from the MR developer community and raise awareness of other

[*]e-mail: sluo5@gmu.edu
[†]e-mail: xinyuhu01@gmail.com
[‡]e-mail: zyan4@gmu.edu

potential threats in MR.

Realizing the attack system requires overcoming two unique challenges in MR. *First*, prior keystroke inference on VR and smartphones [7, 20, 27, 36] only analyze three degree-of-freedom (3DoF) device motion to infer user input whereas HoloLens motion in MR has six degree-of-freedom (6DoF). Specifically, virtual keyboards in smartphones or VR systems are moving along with the device by following user hand or head movement. However, once a HoloLens virtual keyboard is activated, it will be fixed in the physical world unless the user finishes entering the data and closes the keyboard. During a key-entering session, the keyboard will not follow the user/HMD as the user steps away from the keyboard (see the supplementary video submission). Therefore, there exists 6DoF device motion in MR, i.e., both HMD orientation and location. The same head orientation may point to different keystrokes while different head locations may result in the same keystroke. *Second*, smartphones or VR headsets generate an explicit motion signal, such as clicking the VR controller or tapping a button on the smartphone, which identifies the moment of a keystroke. But no such explicit indication exists in the motion of an MR headset. A user makes keystrokes in MR by performing a finger-tapping gesture with her hand in front of the HMD camera, known as the "air tap" [5]. This key-committing process does not incur any obvious device motion and thus leaves no explicit indication of when a key is selected.

In this paper, we bridge the aforementioned gaps and expose the first malware-based keystroke inference attack in MR. Our attack system, namely HoloLogger, enables 6DoF device motion tracking in the 3D space. By recording both HMD orientation and location, we map the 6DoF HMD motion[1] collected by the malware to the target keys on the MR keyboard. Furthermore, HoloLogger identifies the keystroke moments by exploring the unique timing pattern of the HMD motion during an "air tap". We propose a cursor-speed-based segmentation algorithm to differentiate the boundaries between when users browse the keyboard and when users actually commit the keystroke. We also propose a time-length-based recognition algorithm to identify the repetitive input of the same key.

We evaluate the performance of HoloLogger with extensive experiments. Through our evaluation involving 25 users and 750 sensitive inputs where passwords consisting of 4–8 lowercase English letters are considered, we validate the accuracy and robustness of HoloLogger. The top-1,3,5 accuracy of the overall keystroke inference reaches 73%, 89%, and 93%, respectively. We also evaluate HoloLogger under different practical environments such as different password lengths, password types, user positions, sampling rates, implementations of system modules, and HMD wearing time. The results suggest that HoloLogger maintains a promising inference accuracy in these dynamic environments, proving its effectiveness and the importance of exposing this security threat.

Our contribution in this paper is summarized as follows.

- We expose the App permission vulnerability and launch the first malware-based keystroke inference attack in MR systems.
- We design several algorithms in the attack framework to analyze the 6DoF head motion for keystroke inference.
- We perform an extensive evaluation of the HoloLogger prototype including evaluating the impact of segmentation, user movement, password length, etc.

2 Related Work

Malware-based Attacks. Traditional malware-based attacks collect user's motion data including hand or head movement to achieve the attack on physical keyboards such as point-of-sale(PoS) machines [37] and computer qwerty keyboards [14], as well as virtual keyboards on smartphones [7, 9, 27, 36, 37] and VR HMDs [20]. For example, WristSpy [9] presents an attack on smartphones by using smartwatch motion data to collect hand motion. Ling *et al*. [20] collect user's head movement from a VR HMD and infer the input made on VR virtual keyboard. However, these prior studies utilize 3DoF motion data that embeds explicit signals of key-committing moments, making them inapplicable to MR which entails 6DoF motion data without explicit keystroke indication.

Video-based Attacks. Video-based attacks extract information from the video recordings of a user typing session. The recording could be direct surveillance [18, 34], reflective surfaces [4], user's eye movement [10], tablet backside [33], fingertip movements [38], or hand clicking action [20] during the user's input process. For example, EyeTell [10] records the user's eye movement using the front camera on the smartphone and infers the input made on the touch screen. Nevertheless, the video-based attacks require an additional camera placed near the victim, which is not practical in a private environment and could raise suspicion in a public environment. Instead, HoloLogger does not require a camera or any other additional hardware to be deployed by the user.

Wireless Signal-based Attacks. Wireless signal-based attacks [1–3, 8, 19, 32] extract the channel stats information from WiFi signal revealing its turbulence caused by the user's hand movement to eavesdrop keystrokes. For example, VR-Spy [2] detects user's gestures while navigating the VR controller and predicts the corresponding committed keystrokes using a k-nearest neighbor classifier. In addition to the burden of deploying extra senders and receivers near the victim, the attack accuracy highly relies on the environment. For example, the surrounding must be controlled to ensure that movement is only generated by the victim while typing the keyboard. In contrast, HoloLogger is more resistant to surrounding turbulence and is more practical in real-world environments.

3 HoloLens Background

Head Gaze and Commit Input Model. Head gaze and commit (HGC) input model is the foundation of the HoloLens interaction system [23]. When a user moves her head around using HoloLens, the cursor on the virtual screen approximates where the user is looking at. To commit a keystroke, a user performs a particular hand gesture called "air tap" in front of the HMD camera. The user first raises the index finger straight up towards the ceiling and then pinch the index finger with the thumb and release the index finger. Once the HMD camera detects an air tap, the cursor will be hit in the current position to commit a keystroke. HGC has been the most popular input method in MR since it resembles the way we interact with computers via a cursor, i.e., moving a cursor around and using one's hand to commit the selection [13, 22, 35]. It is used by default in HoloLens 1 and is also supported by HoloLens 2 and other MR devices. Therefore, we focus on HGC in this paper.

HoloLens Coordinate System. An illustration of the geometrical relationship among HMD and virtual objects in the HoloLens coordinate system [17] is shown in Figure 2. The origin point is created by each App on startup and is fixed at the initial HMD location through the App's entire lifetime. HoloLens system keyboard is a planar object rendered 1 meter away relative to the HMD location and centered at the direction of the head gaze in the coordinate system. It is a world-locked content, indicating that it is fixed in the physical environment once rendered. Its location may only change when the user finishes the current input, closes the keyboard, and reactivates it in a different direction. Our attack focuses on one key-entering session while the keyboard is open and fixed in order to simulate the real-life scenarios where a user enters a password for one account. The system keyboard used across Apps is a "QWERTY" keyboard.

Motion Data Source. In HoloLens, the 6DoF head motion is defined by the coordinates of head location and orientation [24]. As the MR interface is navigated through by head movement, all Apps need

[1] As HMD motion is driven by head motion, we use them interchangeably.

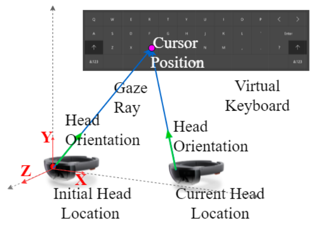

Figure 2: HoloLens coordinate system.

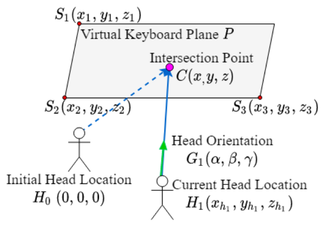
$S_1(x_1, y_1, z_1)$
Virtual Keyboard Plane P
Intersection Point $C(x, y, z)$
$S_2(x_2, y_2, z_2)$
$S_3(x_3, y_3, z_3)$
Head Orientation $G_1(\alpha, \beta, \gamma)$
Initial Head Location H_0 (0, 0, 0)
Current Head Location $H_1(x_{h_1}, y_{h_1}, z_{h_1})$

Figure 3: Cursor position derivation.

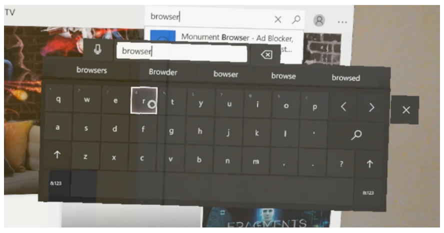

Figure 4: The screenshot of the default HoloLens keyboard.

to constantly read the head motion. Therefore, no user authorization is required for MR Apps to access the head motion data. We utilize this vulnerability to enable our keystroke inference attack.

4 Threat Model

We assume a scenario where a user is entering sensitive information using the HGC input model in MR. For example, the user enters a password for the login of her email or social media account. We consider users who are inside private areas such as their own offices. Since any extra signal harvesting hardware will be noticeable in these areas, prior hardware-assisted attacks [1, 20] no longer work.

The attacker's goal is to infer the user's sensitive input data by deploying malware. We assume the attackers cannot possess or approach the HMD and then deploy the malware directly. However, the attackers may develop a seemingly benign App that embeds the malicious code, such as a regular game App or a music player, and publish it in the official App Store. Then users may choose to install the App by themselves or the attacks may trick the users to install that App for gaming or playing music. This type of malware deployment is difficult to detect as users can still enjoy the normal functions of the malicious App. This deployment method has been widely reported for a decade [11, 40, 41]. As HoloLens is an off-the-shelf MR HMD and provides a universal virtual keyboard across its Apps by default (see the submitted video), the attacker can obtain the layout of this default virtual keyboard before launching the attack.

We consider a default HoloLens operating system available to users without requiring the research or root mode. Through official HoloLens APIs, the malware can detect the activation and closing of the virtual keyboard and collect HMD motion data in between. It can then send the data to attackers for keystroke inference. The entire process does not require user permission since access to motion data and network are by default allowed in MR Apps.

Note that we do not consider the situation where the malware in the malicious App can directly extract keystrokes from a victim App because this assumption would be too strong. In the victim App, keystrokes are stored in local variables. Directly extracting keystrokes from the App would require the malware to access the victim App's local variables in runtime, which is impossible without modifying the victim app. Instead, HoloLogger utilizes the `head-gaze` API to monitor the user head motion that is enabled by default for all Apps [25], which does not require accessing the data of the victim App.

5 6DoF-Head-Motion-driven Key Tracking

As depicted in Figure 1, the first module of HoloLogger is a 6DoF head motion driven key tracking module. The original head motion data logged in the malware is a stream of 6DoF samples, including two coordinates representing the head location and orientation. By using the proposed key tracking scheme in this section, the head motion stream will be translated into a stream of cursor positions which will then be mapped to a stream of keys aimed at by the user.

5.1 Head to Cursor Projection

As a user can move around while typing, the distance and direction between the user and the virtual keyboard will change. Figure 3 represents the geometry relationship between the varying user head location H_0, H_1 and the fixed virtual keyboard plane P. The initial head location that is determined when the app is launched is designated as the origin point $H_0(0,0,0)$. The current head location $H_1(x_{h_1}, y_{h_1}, z_{h_1})$, head orientation $G_1(\alpha, \beta, \gamma)$ are obtained for each head motion sample. The three boundary vertices $S_i(x_i, y_i, z_i), i = 1,2,3$ (top left, bottom left and bottom right) of the keyboard are obtained when the keyboard is rendered. Identifying the cursor position on the virtual keyboard is a line–plane intersection problem [31]. It can be solved by the following steps.

The normal vector $\overrightarrow{n_s}(x_n, y_n, z_n)$ of the keyboard plane P can be found by the cross product of two vectors on P. Given the coordinates of S_1, S_2, S_3, we have the following formal determinant

$$\overrightarrow{n_s} = \overrightarrow{S_1 S_2} \times \overrightarrow{S_2 S_3} = \begin{vmatrix} \overrightarrow{i} & \overrightarrow{j} & \overrightarrow{k} \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_2 & y_3 - y_2 & z_3 - z_2 \end{vmatrix} = (x_n, y_n, z_n) \quad (1)$$

where $S_1(x_1, y_1, z_1)$, $S_2(x_2, y_2, z_2)$ and $S_3(x_3, y_3, z_3)$ are the top left, bottom left and bottom right vertices of the keyboard. $\overrightarrow{i}$, $\overrightarrow{j}$ and $\overrightarrow{k}$ are the standard basis vectors of the HoloLens coordinate system.

Suppose the intersection point between the keyboard plane and the gaze ray is $C(x, y, z)$. As a vector defined by two points in a plane is within that plane, $\overrightarrow{S_1 C}$ is within the keyboard plane. Since the dot product between the normal vector of a plane and any vectors within that plane is zero, we have $\overrightarrow{n_s} \cdot \overrightarrow{S_1 C} = 0$.

Since the line of gaze ray is overlapped with the head orientation, the line equation of the gaze ray can be found by head location and head orientation using the following equation,

$$\frac{x - x_{h_1}}{\alpha} = \frac{y - y_{h_1}}{\beta} = \frac{z - z_{h_1}}{\gamma} \quad (2)$$

Therefore, given the collected head location $(x_{h_1}, y_{h_1}, z_{h_1})$ and head orientation (α, β, γ), solving equation (1)-(2) gives the cursor position $C(x, y, z)$.

5.2 Target Key Identification

Once the cursor position is available, the corresponding key can be identified by comparing the cursor position with each key's boundary. As mentioned in Section 3, the virtual keyboard is fixed during an input session, e.g., entering a password. The keyboard width and height can be derived from boundary vertices S_1, S_2, S_3. To get the boundaries of each key, the keyboard layout of the HoloLens operating system is needed. This is the system default keyboard employed across Apps such as App Store, Browser, and Skype. As shown in Figure 4, there are 4 rows of keys on the keyboard. Except for the space key and the enter key, all other keys occupy just one cell. Based on this layout, the vertices and boundaries of each key can be easily calculated.

The cursor position and key boundary information are both derived in the HoloLens coordinate system with the basis vectors

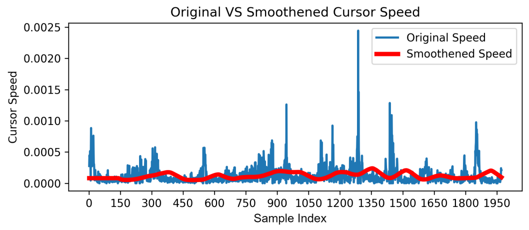

Figure 5: Removing background noise caused by heavy HMD.

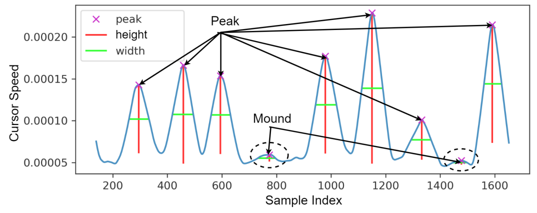

Figure 6: Peaks, not mounds, must be identified to find low-speed segments (valleys).

$\overrightarrow{i}, \overrightarrow{j}, \overrightarrow{k}$. However, cursor position and keyboard boundary are all points in the keyboard plane (see Figure 2 and 3). To easily find the corresponding key for a cursor position, we apply change of basis to convert all coordinates from the HoloLens coordinate system to the keyboard plane coordinate system defined by $\overrightarrow{i'}, \overrightarrow{j'}, \overrightarrow{k'}$ so that $\overrightarrow{k'}$ axis can be eliminated.

We need to derive the transformation matrix A to conduct the coordinate conversion which only includes translation and rotation. First, the origin of the keyboard plane coordinate system is at point S_2 with positive X-axis directing to point S_3, the positive Y-axis directing to point S_1 and the positive Z-axis pointing towards the user. We know the transformation result is the basis vectors in *keyboard plane coordinate system* $\overrightarrow{i'} = (1,0,0)$, $\overrightarrow{j'} = (0,1,0)$, $\overrightarrow{k'} = (0,0,1)$. In *HoloLens coordinate system*, suppose $\overrightarrow{i'} = (x_{i'}, y_{i'}, z_{i'})$, $\overrightarrow{j'} = (x_{j'}, y_{j'}, z_{j'})$, $\overrightarrow{k'} = (x_{k'}, y_{k'}, z_{k'})$. $\overrightarrow{i'}, \overrightarrow{j'}, \overrightarrow{k'}$ can be obtained by normalizing the keyboard plane vectors $\overrightarrow{S_2S_3}, \overrightarrow{S_2S_1}$, and their normal vector $\overrightarrow{S_2S_3} \times \overrightarrow{S_2S_1}$:

$$\begin{bmatrix} \overrightarrow{i'} & \overrightarrow{j'} & \overrightarrow{k'} \end{bmatrix}^T = \begin{bmatrix} x_{i'} & x_{j'} & x_{k'} \\ y_{i'} & y_{j'} & y_{k'} \\ z_{i'} & z_{j'} & z_{k'} \end{bmatrix}^T = \begin{bmatrix} \frac{\overrightarrow{S_2S_3}}{|\overrightarrow{S_2S_3}|} & \frac{\overrightarrow{S_2S_1}}{|\overrightarrow{S_2S_1}|} & \frac{\overrightarrow{S_2S_3} \times \overrightarrow{S_2S_1}}{|\overrightarrow{S_2S_3} \times \overrightarrow{S_2S_1}|} \end{bmatrix}^T \tag{3}$$

Then the change of basis can be expressed as $A \cdot \begin{bmatrix} \overrightarrow{i'} & \overrightarrow{j'} & \overrightarrow{k'} \end{bmatrix}^T = I$. Thus $A = (\begin{bmatrix} \overrightarrow{i'} & \overrightarrow{j'} & \overrightarrow{k'} \end{bmatrix}^T)^{-1}$.

Finally, we can transform any coordinate (x,y,z) in the HoloLens coordinate system to (x',y',z') in the keyboard plane coordinate system via $\begin{bmatrix} x' & y' & z' \end{bmatrix}^T = A \cdot \begin{bmatrix} x & y & z \end{bmatrix}^T$.

By comparing transformed coordinates of cursor positions with the key boundary, we can find the target key. As a result, the stream of head motion is translated into a stream of target keys.

6 Air-Tap-Pattern-based Keystroke Inference

In this section, we infer the keystrokes from the stream of cursor positions and target keys. First, by analyzing the unique behavior pattern when users conduct an air tap, a cursor speed-based segmentation algorithm is proposed to divide the target key stream into segments of potential keystrokes. These potential keystrokes, namely *base candidate*, will undergo a repetitive keystroke recognition algorithm to determine whether or not an identical key has been hit more than once in the segment. Then the keystroke inference result, i.e., the top-1 candidate, will be generated. Lastly, an inference calibration is conducted to address head motion fluctuation and generate more candidates for the attack.

6.1 Cursor Speed-based Segmentation

There are two types of cursor moving patterns in the collected stream: browsing and typing. Browsing is the high-speed motion when a user is looking for the target key. Typing is the low-speed motion when the user finds the target key and stabilizes the cursor to commit the keystroke. Only typing motion is needed to infer user input. To differ typing from browsing, the cursor moving speed is calculated by subtracting the previous cursor position from the current cursor position. This is because the cursor positions are collected at a fixed sample rate and the displacement is equivalent to the speed.

The first step of the segmentation is to remove noise and clean the cursor speed data. Due to the fact that MR HMD is heavier than handheld devices, head motion is observed to contain more noise than hand motion. There is continuous background noise caused by small head movement. It might not be noticeable by humans but exists in the entire data stream. To remove this noise, second-degree Simple Moving Average is applied to smooth the cursor speed data. As shown in Figure 5, the short-term fluctuations in the original speed data are eliminated but the cursor moving trend is kept.

Since the typing motion is associated with lower cursor moving speed compared to the browsing motion, the segmentation problem can be translated into a valley finding problem in the cursor speed data. We solve the valley finding problem by first identifying peaks because peaks contain consistent features that are easier to be distinguished from valleys. For instance, valleys can have various sizes of fluctuations but peaks have more static prominence which can be used for easy identification. The peaks can be identified by finding local maximum, which are defined as any samples whose two direct neighbours have a smaller amplitude.

However, identifying peaks by local maximum may lead to false peaks that are actually mounds. A mound is motion noise caused by user overshooting or glancing while typing one key. The comparison of peak and mound is shown in Figure 6. If a mound is identified as a peak, it will break one segment into two. The second valley would become the false valley which produces an extra wrong key in the final keystroke candidate. Therefore, mounds should not be identified as peaks and should be viewed as a part of the valley. To address this issue, our segmentation algorithm utilizes peak width and height based thresholds to filter out mounds. Once all the peaks are identified, valleys can be found by taking out the peaks from the stream and keeping the remaining segments.

The output of the segmentation algorithm are the start and end indexes of all segments. In each segment, HoloLogger records multiple samples of cursor position and targeted key. Since the user would spend most of the time within one typing segment aiming at the committed key, we apply majority voting within a segment to convert samples of target keys to the one single keystroke that appears the most frequently. The single keystroke from all segments become the *base candidate* for the keystroke inference.

6.2 Repetitive Keystroke Recognition

Repetitive keystroke occurs when the user first browses the keyboard to locate the target key and then air taps the key for multiple times. The motion difference among committing these repetitive keystrokes only exists in hand motion rather than head motion. The cursor speed during the repetitive keystrokes is minimal. Therefore, the head motion for committing repetitive keystrokes appears to be similar to the motion when committing one keystroke, making the repetitive keystrokes difficult to recognize.

Based on the fact that repetitive keystrokes take longer time to enter, we recognize the keystroke within a segment of a longer length as the repetitive keystroke. We focus on addressing repetitive

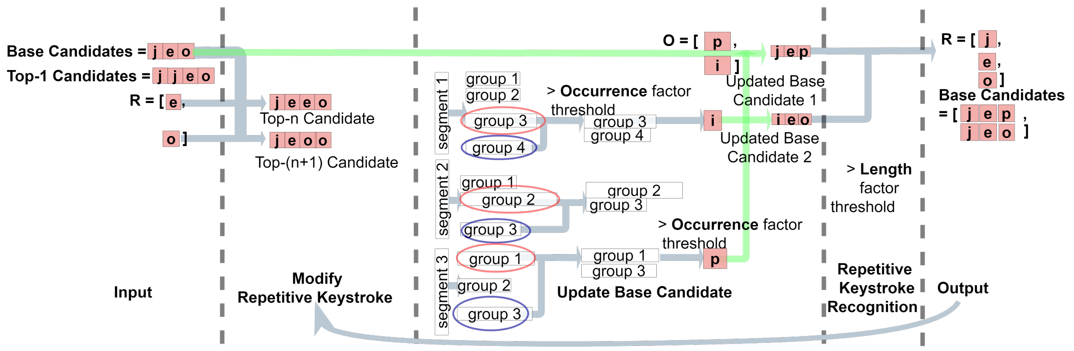

Figure 7: Overview of the inference calibration algorithm.

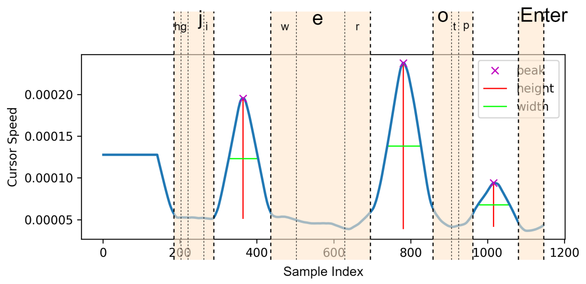

Figure 8: Repetitive keystroke recognition by segment length.

keystroke that contains two identical keystrokes. However, the proposed segment time length-based algorithm is generic and can be applied to cases that repeat the keystroke three times or more. From the collected data, we observe that repetitive keystrokes take longer than once but less than twice of the time spent on striking one non-repetitive key because browsing is skipped for selecting the second key. As shown in Figure 8, each segment's longest keystroke "j", "e", "o") is selected to compose the base candidate ("jeo") despite the browsing on some other keys (e.g., cursor on "w" and "r" when striking "e"). Since the segment containing "e" lasts longer than others ("j" and "o"), it is likely to contain repetitive keystroke. To determine whether a segment has repetitive keystrokes, we define keystroke length factor as $\frac{N_p}{N_{\text{avg}}}$, where N_p is the length of the selected keystroke in p^{th} segment, N_{avg} is the average keystroke length and the unit is the number of samples. Note that if the user commits keystrokes on adjacent keys (e.g., "wer"), she would browse to the target key, hold her hand to air-tap, and repeat this for all keystrokes. Thus Figure 8 would have three separate segments corresponding to "w", "e" and "r", which can be directly recognized.

The length factor of each keystroke is calculated. If the length factor of a keystroke is above a threshold, we add this keystroke to the potential repetitive keystroke list R. If there are multiple keystrokes in R, they are ranked by the values of length factor. The highest-ranking keystroke will be used to form the inference result. The inference result will repeat this recognized keystroke for two times. If there is no keystroke with a length factor above the threshold, the inference result will be the same as base candidate generated in the segmentation step. For example, if only keystroke "e" is found with length factor above the threshold in Figure 8, "e" is added to the repetitive keystroke list R, and the inference result will contain "ee" instead of "e".

6.3 Inference Calibration

To improve HoloLogger's performance, we conduct several steps to calibrate the above inference result (the top-1 candidate), and generate more candidates for the keystroke inference. The calibration algorithm is illustrated in Figure 7 and explained as follows.

Calibration 1 – Modify Repetitive Keystroke. As MR interaction behavior varies across different users and input attempts, it is possible that the longest segment may not be the one having repetitive keystroke. For example, when a user has trouble aiming at the target key and spends a long time stabilizing the cursor, the segment would be long but actually include one keystroke. Hence, a desired calibration method is to modify repetitive keystroke.

First, we remove the repetitive keystroke, if any, from the top-1 inference result. This way, the candidate is restored to the base candidate. We then check the potential repetitive keystroke list R generated in Section 6.2. As the highest-ranking keystroke in R has been used in top-1 candidate, we select the next highest-ranking keystroke in R and repeat this keystroke twice to form the calibrated result. This step is repeated until all keystrokes in R have been tried. In the example shown in Figure 7, R begins with two keystrokes "e" and "o". Thus, from R and the base candidate, we can built two more candidates: "jeeo" and "jeoo".

Calibration 2 – Update Base Candidate. In addition to incorrect inference of repetitive keystroke, it is possible that the base candidate obtained in Section 6.1 needs revision. Recall that the recognized keystroke of each segment is selected by majority voting. However, it is possible that the most frequently appeared key is not the actual committed key. A typical scenario is when the user is trying to commit a keystroke but spends the same or even longer time looking at a nearby key than the key to strike. In such cases, the majority voting fails to recognize the actual keystroke in that segment.

To address the incorrect base candidate, we propose to further calibrate result since the correct key is also recorded in the target key stream. We further divide a segment into different groups of samples aiming at different keys. We define the *occurrence* of a key group as how many samples are collected during the aiming. The group with the highest occurrence has been used for the original base candidate. For calibration, we find the second most frequently-aimed key and compare its occurrence to the most frequently-aimed key in order to see whether it is the missed keystroke. The closer their occurrence are, the more likely that the second most frequently-aimed key is the missed keystroke. Thus, we define the occurrence factor as $\frac{M_q}{M_{\max}}$, where M_q is the occurrence of the q^{th} key group and $M_{\max}$ is the occurrence of the most frequently-aimed key in the same segment.

The occurrence factor of each key group is calculated. If the occurrence factor of a key group is above a threshold, its corresponding key is added to the potential missed keystroke list O. If there are multiple keys in this list, they are ranked by the values of occurrence factor. The highest-ranking key will replace the original key of that segment to form the updated base candidate. The remaining keys in list O will be used to generate more updated base candidates in the same way. For each updated base candidate, we loop back to the step in Section 6.2 and Calibration 1 again to generate corresponding candidates. An example is shown in Figure 7. We first find the second longest key group in each segment. By calculating their key groups' occurrence factors, we know segment 1 and 2's keystrokes could be updated. Hence, the keystrokes of these groups are added

Figure 9: Experiment setup.

to O. Then, from the current base candidate “jeo” and O, we can update the base candidate by substituting the original keystroke with the one in O. This step is repeated until all keys in O are tried. In this way, we can generate the updated base candidates “jep” and “ieo”. On top of these updated base candidates, we may repeat repetitive keystroke recognition, Calibration 1 and 2 until a desired number of top-k candidates is generated.

7 Experiments

Apparatus. As shown in Figure 9, the testing environment of HoloLogger consists of a HoloLens version 1 and a laptop. The laptop for data processing is a ThinkPad P71 with a 3.10GHz CPU. This simulates the threat model where data is processed in a backend system.

On HoloLens, malware is implemented with Unity game engine and disguised in a benign App. It listens to the keyboard invoking event from HoloLens API to capture the keyboard activation time and closing time. The head motion is logged at 70 fps while the keyboard is activated. In addition, the three boundary vertices of the keyboard are acquired on the keyboard's startup. The data will be sent to the laptop when the network is available. We employ popular Python libraries including Pandas and Numpy for data processing.

Participants. A total of 25 participants (9 females and 16 males, ages 22 to 32) were recruited through public advertisements. All participants were university students. Among the participants, 8 of them wore glasses and they kept their glasses on during the experiments. Ten participants have used MR more than once before and 15 never used it. Since we did not observe a significant difference in attack performance in different user groups, we do not separate the results.

Participants signed a written consent form in accordance with existing IRB approval which allows us to record head navigation and location data from human subjects for discovering the vulnerability of MR platforms. A brief introduction of Hololens interaction was given to the participants. To help participants get familiar with Hololens operation, they were first asked to take a training session, during which they tried five 8-character inputs. The experiment only started when the participants felt familiar with the interaction. Since the goal of HoloLogger is to infer sensitive keystrokes, we focus on hacking user passwords in the evaluation. Note that whether a “username” is associated with the password does not bias the evaluation results. During the experiment, a participant sat in a relaxed posture wearing the HMD and typed a set of given passwords picked from the list of 100 most common passwords [30] to validate the performance of HoloLogger in realistic situations. The set includes 30 passwords with lengths ranging from 4 to 8 lowercase letters (each length category contains 6 passwords), half of which contain repetitive keystrokes. A total of 750 password trials were collected. We focused on English passwords without numbers and special symbols because entering them requires switching to another virtual keyboard with a different layout (see Figure 4).

Evaluation Metrics. We use F1 score to evaluate the performance of keystroke segmentation. The ground truth is the list of passwords we instructed users to input. If an identified segment (valley) has an overlap with one of the ground truth segments, it is counted as the correct segment.

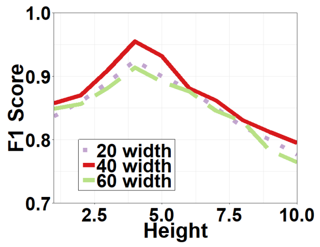

Figure 10: F1 score with different peak width and height thresholds.

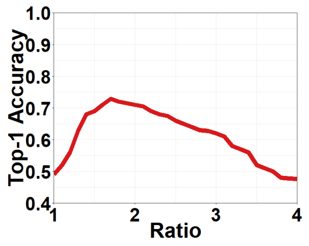

Figure 11: Top-1 accuracy with different length factor values.

We also use the hit rate of top-1,3,5 candidates generated by HoloLogger to evaluate the overall keystroke inference performance. A password is hit when it exactly matches the inference result.

$$Inference\ Accuracy = \frac{No.\ of\ Matched\ Passwords}{No.\ of\ All\ Passwords} \times 100\% \quad (4)$$

8 Evaluation Results

In this section, we evaluate the performance of HoloLogger. We start with a micro-benchmark to perform sensitivity analysis of system parameters and evaluate components of HoloLogger. We then conduct a set of test attacks under various practical environments, such as different password lengths, password types, user movements, sampling rates, implementations of HoloLogger modules, and HMD wearing time. Finally, we present the results of a user experience study to obtain more insights of MR keystroke inference.

8.1 Micro-benchmarks

In this section, we aim to adjust the parameters in HoloLogger and optimize each module. Note that to avoid bias, the data used in this section is separated from the data used in Section 8.2. It includes 5 random passwords with lengths ranging from 4 to 8 characters for each user (125 password trials in total).

Optimizing Segmentation. Recall that we use peak width and height thresholds to filter out mounds and avoid the introduction of incorrect segments. To evaluate the performance of this threshold-based segmentation, we present the results of F1 score under different width thresholds and height thresholds in Figure 10. Under a fixed height threshold, we observe that as the width threshold increases, the accuracy increases continuously, reaches a maximum, and then decreases gradually. This is because when the width threshold is low, most of the mounds are above the threshold and identified as peaks. These false peaks lead to false valleys and thus low accuracy. As the width threshold approaches the width of most mounds, more mounds are correctly filtered. However, when the width threshold becomes too high, even actual peaks are not wider than it. Then multiple valley segments are falsely identified as one single segment. As a result, multiple keystrokes are missed in the segmentation.

The maximum accuracy, 96%, is achieved when the width threshold is 40 samples and the height threshold is $3.8 \times 10^{-6} m/s$. We keep these values to maintain the best segmentation performance in the remaining tests.

Optimizing Repetitive Keystroke Recognition. To ensure repetitive keystroke is correctly recognized, we design a length factor in Section 6.2 to identify unusually long segments. In this section, we show the inference performance under varying values of the length factor. As shown in Figure 11, the top-1 inference accuracy increases to a maximum of 75% while the length factor increases to 1.7. Then the accuracy drops as the length factor further increases. This is because if the length factor is too small, a single keystroke is recognized as repetitive keystrokes whereas if the length factor is too

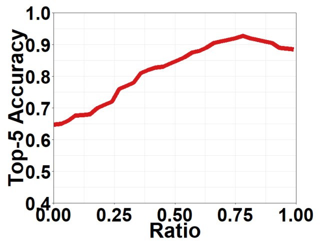

Figure 12: Top-5 accuracy with different occurrence factor values.

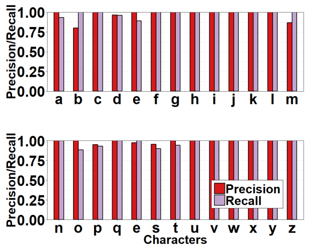

Figure 13: The system achieves promising performance across all characters.

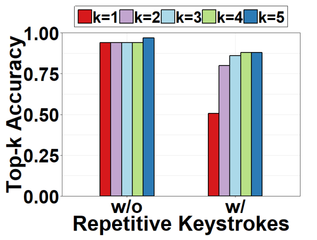

Figure 14: The system accurately identifies repetitive keystroke.

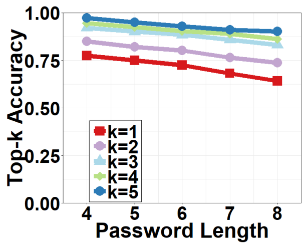

Figure 15: Inference results of passwords of different lengths.

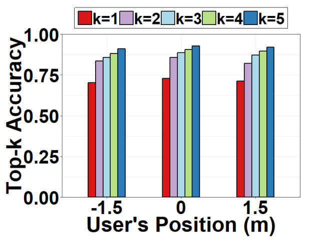

Figure 16: User's positions do not affect top-k inference accuracy.

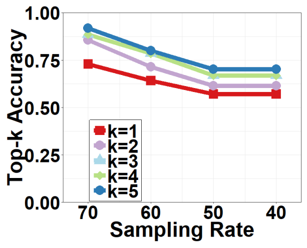

Figure 17: The system achieves acceptable performance given limited sampling rates.

large, actual repetitive keystrokes are not found. Both Scenarios will result in wrong password inference. Therefore, the optimal length factor is set for implementing HoloLogger.

Optimizing Inference Calibration. As the base candidate may be derived incorrectly, the inference calibration is important to improve the top-k inference accuracy. We evaluate the accuracy of the inference calibration results, i.e., the top-5 candidates under different values of occurrence factor. Results are shown in Figure 12. While the occurrence factor is increasing, the accuracy increases as well. It can be explained as that with a low occurrence factor the correct base candidate will be updated to the next-frequently-aimed key in the same segment even if the aiming time for that key is much lower than the correct key, which decreases the accuracy. The optimal accuracy, 94%, is achieved when the occurrence factor is 0.78.

8.2 Keystroke Inference Evaluation

In this section, we investigate the performance of HoloLogger under various practical environments. To avoid bias, the data used in this section are 625 new passwords different from the previous tests.

Impact of Different Characters We evaluate the keystroke inference performance for each individual English character on the keyboard. This serves as the benchmark for the multi-key password inference. In this evaluation, HoloLogger only executes the processing until segmentation. No repetitive keystroke recognition or calibration is performed. The precision and recall of inferring 26 English characters are shown in Figure 13. Based on the result, we conclude that HoloLogger achieves promising performance across all characters and the challenges of keystroke inference come mainly from multi-key cases.

Impact of Repetitive Keystroke We also divide the passwords into two groups based on whether they contain repetitive keys and compare the inference accuracy between them. The top-k accuracy of passwords in the complete set as well as in two groups is shown in Figure 14. While the top-1 accuracy is low for passwords with repetitive keystrokes, the accuracy of cases with more attempts, e.g., top-5, is only slightly impacted. This is because the inference calibration process effectively identifies most of the repetitive keystrokes within the first five trials. This result proves that HoloLogger is able to accurately identify repetitive keystrokes under several trials. We also observe that the top-1 to top-4 accuracy for inferring passwords without repetitive keystrokes are the same while the top-5 accuracy increases around 2%. The improvement appears only in the fifth attempt because inference errors are largely caused by identifying a single long-lasting keystroke as two repetitive keystrokes. Thus Calibration 1 in Section 6.3 does not help in the first four attempts. The accuracy only improves when Calibration 2 kicks in and updates the base candidate.

Impact of Password Length Users may set up passwords or sensitive information with different lengths. We evaluate the impacts of the length of input on the inference accuracy. The number of characters in our test input ranges from 4 to 8. We calculate their top-k inference accuracy and the results are summarized in Figure 15. It shows that the accuracy is impaired by the increase of password length. The reason is that longer passwords have a higher chance of wrong segmentation and repetitive keystroke recognition. But it is also important to note that the longer password length is not sufficient to prevent the attack. For example, the top-5 accuracy drops from 97% to 90% when the password length increases from 4 characters to 8 characters.

Impact of User Movement As HoloLogger tracks head motion in 6DoF, a user's positional change, i.e., moving from one place to the other, should not affect the attack performance. We conducted a sub-test to validate this idea. During this test, 5 users were asked to input 30 passwords at 3 different positions including the initial position when the App is started, as well as 1.5 meters to the right and left from the initial position. A total of 450 passwords were collected. In Figure 16, we show the results when data is collected at the initial position, 1.5 meters to the right and left from the initial position, respectively. We observe that such head location change does not affect the inference accuracy. This is because HoloLogger infers the keystroke through 6DoF motion data. This result verifies the effectiveness of HoloLogger under varying input locations.

Impact of Sampling Rates The sampling rate of the HoloLogger system is an important parameter. In this section, we evaluate how the sampling rate of the malware will affect inference accuracy. To study this, we use different amounts of collected data to simulate different sampling rates at 40, 50, 60, and 70 fps (primary), respectively. The inference results are shown in Figure 17. We observe that the accuracy decreases as the sampling rate decreases. This is because less data leads to more segmentation and recognition errors. Therefore, a reasonable sampling rate is required to conduct the keystroke inference.

Importance of Inference Calibration To investigate to what extent the inference calibration process improves the inference results, we implement different variants of HoloLogger by disabling different designs within this process and evaluate the variation of the accuracy. We first disable the repetitive key recognition (denoted as RC). Then we activate the repetitive character recognition function and disable the base candidate updating design (denoted as BC). We compare these two variants with the full HoloLogger (denoted as Complete) and show the results in Figure 18. We observe that the top-1,3,5 accuracy of complete HoloLogger is 73%, 89%, and 93%

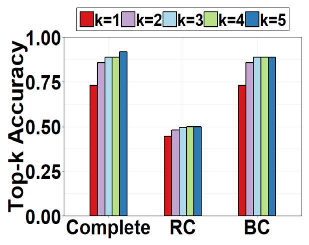

Figure 18: Repetitive keystroke recognition improves accuracy.

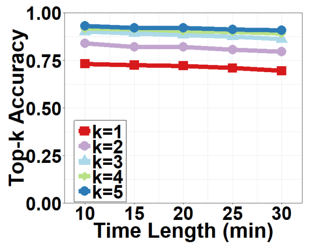

Figure 19: Inference results under different HMD wearing time.

respectively. In addition, the repetitive key modification drastically improves the system performance. This indicates that handling repetitive keystrokes is a significant factor for the keystroke inference to succeed.

Impact of HMD Wearing Time We also study if factors related to users would affect the inference accuracy. Specifically, we are interested in the impact of user fatigue. Since HoloLens is a heavy HMD, user behavior may change dramatically when they wear the HMD for a long time. A sub-test is conducted with 5 participants. Each participant is asked to wear the HMD for different time periods before starting the typing experiments. The result in Figure 19 shows top-k accuracy slightly decreases when the wearing time increases. This may be attributed to the fact that when a user wears the HMD for a longer time, it is harder to hold HMD steadily to aim at the key. However, the accuracy decrease is negligible. This is because HoloLogger can handle different types of motion noise. Thus, this result validates the generality of HoloLogger.

8.3 User Experience

We collect user feedback on HoloLogger through a user experience survey for a set of comparison experiments. In the experiments, users type passwords in an App with the malware and an identical App without the malware. Users do not know the difference between these two Apps. Users also try both HGC and voice input. All 25 users attend the survey by completing a questionnaire immediately after the experiment. The questionnaire asks users to compare the malware with the benign App in responsiveness. In addition, the survey asks about user opinion of the HGC compared to the voice input on several aspects: convenience, security, and social comfort (comfortable to use in the public). The survey result is summarized in Table 1. Most of the participants did not experience significant lagging. This suggests that no suspicion will be raised about this malware in terms of responsiveness. We also observe that HGC is more convenient, secure, and acceptable by users. In other words, people prefer HGC rather than voice input in practice which highlights the importance of exposing this threat of keystroke inference.

9 Discussion and Future Work

Generalization for Keyboard Layout and MR Devices. In this work, we implemented HoloLogger based on the parameters of the default HoloLens keyboard. Although this keyboard has been used for most HoloLens Apps, we acknowledge that some Apps may adopt a customized keyboard layout and that the system-default layout may change upon operating system update. However, the proposed designs of HoloLogger are generic. Attackers can still obtain the updated layout information by downloading the target OS and the target App. They can then update the keyboard parameters in the derivation in Section 5. Therefore, HoloLogger can be extended to accommodate various types of keyboards.

Similarly, HoloLogger can be extended to other MR devices that support the HGC input model than the HoloLens 1. HoloLogger is designed based on the user interaction pattern in the HGC model rather than specificity in HoloLens 1. By using the keyboard parameters of the target MR device, attackers can launch the attack at any HGC-based device. As HGC has been the default input modality in HoloLens 1 and is still adopted by HoloLens 2, our attack will bring a broad impact on the MR design.

Table 1: User Feedback.

	Questions	Answer: Y	Answer: N
Q1	Have you used an AR device before?	10	15
Q2	Is there noticeable lagging when typing?	3	22
Q3	HGC input is more secure than voice input	21	4
Q4	HGC input shows more social comfort than voice input	19	6
Q5	HGC input is more convenient than voice input	14	9

Diversity of Repetitive Keystrokes. HoloLogger is designed to infer input containing at most one repetitive keystroke. Although this is true for a large number of sensitive inputs, it is possible that the sensitive input contains multiple repetitive keystrokes (such as “eeoo”), repetitive keystrokes composed of three or more repeating characters (such as “eeeo”), or a combination of both cases (such as “eeeoo”). As a result, the accuracy of HoloLogger will be decreased in these cases. Fortunately, a threshold of the length difference among segments can be included to detect segments with repetitive keystrokes that are drastically longer than others. Also, a length factor can be optimized to determine the number of keys in a repetitive keystroke segment. This indeed requires a larger scale of empirical data and a separate study to finalize the design choices.

Countermeasures and Lessons Learned. Based on the extensive results shown in Section 8, we have found some important implications in terms of defending the proposed MR keystroke inference attack. We recommend these countermeasures for MR users and developers in order to prevent the potential leak of sensitive input data. From the user’s perspective, protection can be achieved by using passwords that contain repetitive keystrokes and more characters. Another method is to change the MR interaction to gaze tracking [15], a touchpad [12, 39] or the hand tracking introduced in HoloLens 2. However, this does not eliminate the threat and the large number of HGC users will still suffer from the proposed attack. On the other hand, from the operating system point of view, while it is not practical to forbid an App’s access to head motion data, the operating system could limit the sampling rate of motion data, especially when the user is making inputs on a keyboard. This should not affect other regular MR interactions, but could reduce the risks of keystroke inference. Furthermore, using a keyboard with randomized key arrangements may help. However, virtual keyboards used for entering sensitive data in most computing platforms such as HoloLens, Android, and iOS are still QWERTY keyboards. The reason is that using randomized key arrangements might affect usability [28, 29]. This makes keystroke inference on standard keyboards indeed a realistic and serious security threat.

10 Conclusion

In contrast with MR’s rising popularity, its security is rarely explored. In this paper, we discuss the feasibility of inferring user input on MR HMDs that use the HGC model. We present the first prototype of a malware-based keystroke inference attack towards these HMDs. Specifically, we exploit the 6DoF head motion data to derive the stream of keys targeted by users. By leveraging the unique head navigation pattern during air taps, keystrokes, including repetitive keystrokes, are segmented and recognized. Extensive experiments through 750 attacks towards passwords consisting of 4 to 8 lowercase English letters among 25 participants show that the proposed attack achieves top-1,3,5 accuracy of 73%, 89%, and 93%, respectively. The exposure of this attack will alert MR users, system designers, and MR developers, and inspire them in designing more secure MR systems. The success of HoloLogger shall also call for more exposure and mitigation of security threats in MR and VR in order to establish trustworthiness in immersive computing platforms.

REFERENCES

[1] A. Al Arafat, Z. Guo, and A. Awad. Vr-spy: A side-channel attack on virtual key-logging in vr headsets. 2021.

[2] A. Al Arafat, Z. Guo, and A. Awad. Vr-spy: A side-channel attack on virtual key-logging in vr headsets. In *2021 IEEE Virtual Reality and 3D User Interfaces (VR)*, pp. 564–572. IEEE, 2021.

[3] K. Ali, A. X. Liu, W. Wang, and M. Shahzad. Keystroke recognition using wifi signals. In *Proceedings of the 21st annual international conference on mobile computing and networking*, pp. 90–102, 2015.

[4] M. Backes, T. Chen, M. Dürmuth, H. P. Lensch, and M. Welk. Tempest in a teapot: Compromising reflections revisited. In *2009 30th IEEE Symposium on Security and Privacy*, pp. 315–327. IEEE, 2009.

[5] R. Ban, Y. Hirao, and T. Narumi. Determining the target point of the mid-air pinch gesture. In *2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pp. 492–493, 2021. doi: 10.1109/VRW52623.2021.00128

[6] L. E. Buck and B. Bodenheimer. Privacy and personal space: Addressing interactions and interaction data as a privacy concern. In *2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pp. 399–400, 2021. doi: 10.1109/VRW52623.2021.00086

[7] L. Cai and H. Chen. Touchlogger: Inferring keystrokes on touch screen from smartphone motion. *HotSec*, 11(2011):9, 2011.

[8] B. Chen, V. Yenamandra, and K. Srinivasan. Tracking keystrokes using wireless signals. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 31–44, 2015.

[9] W. Chen, J. Liu, X. Guo, Y. Wang, and Y. Chen. Wristspy: Snooping passcodes in mobile payment using wrist-worn wearables. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 2071–2079. IEEE INFOCOM, 2019.

[10] Y. Chen, T. Li, R. Zhang, Y. Zhang, and T. Hedgpeth. Eyetell: video-assisted touchscreen keystroke inference from eye movements. In *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 144–160. IEEE, 2018.

[11] P. Cheng, I. E. Bagci, U. Roedig, and J. Yan. Sonarsnoop: Active acoustic side-channel attacks. *International Journal of Information Security*, pp. 1–16, 2019.

[12] R. Darbar, J. Odicio-Vilchez, T. Lainé, A. Prouzeau, and M. Hachet. Text selection in ar-hmd using a smartphone as an input device. In *2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pp. 526–527, 2021. doi: 10.1109/VRW52623.2021.00145

[13] G. Evans, J. Miller, M. I. Pena, A. MacAllister, and E. Winer. Evaluating the microsoft hololens through an augmented reality assembly application. In *Degraded environments: sensing, processing, and display 2017*, vol. 10197, p. 101970V. International Society for Optics and Photonics, 2017.

[14] W. He, T. T.-T. Lai, and R. R. Choudhury. Mole: Motion leaks through smartwatch sensors. In *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pp. 155–166. ACM, 2015.

[15] H. Heo, M. Lee, S. Kim, and Y. Hwang. Gaze+gesture interface: Considering social acceptability. In *2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pp. 690–691, 2020. doi: 10.1109/VRW50115.2020.00196

[16] D. Hosfelt, D. Maloney, K. Bye, T. Beck, E. Southgate, and P. Swire. Ethics and privacy in mixed reality. In *2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pp. 1–1, 2020. doi: 10.1109/VRW50115.2020.00290

[17] F. A. Khan, V. V. R. M. K. R. Muvva, D. Wu, M. S. Arefin, N. Phillips, and J. E. Swan. A method for measuring the perceived location of virtual content in optical see through augmented reality. In *2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pp. 657–658, 2021. doi: 10.1109/VRW52623.2021.00211

[18] C. Kreider. The discoverability of password entry using virtual keyboards in an augmented reality wearable: An initial proof of concept. *Southern Association for Information Systems*, 2018.

[19] M. Li, Y. Meng, J. Liu, H. Zhu, X. Liang, Y. Liu, and N. Ruan. When csi meets public wifi: inferring your mobile phone password via wifi signals. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 1068–1079, 2016.

[20] Z. Ling, Z. Li, C. Chen, J. Luo, W. Yu, and X. Fu. I know what you enter on gear vr. In *2019 IEEE Conference on Communications and Network Security (CNS)*, pp. 241–249. IEEE, 2019.

[21] F. Lu. [dc] glanceable ar: Towards an always-on augmented reality future. In *2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pp. 717–718, 2021. doi: 10.1109/VRW52623.2021.00241

[22] Microsoft. eyeorheadgaze. `https://docs.microsoft.com/en-us/windows/mixed-reality/design/gaze-and-commit#eye--or-head-gaze`, 2019. Accessed: 2021-05-01.

[23] Microsoft. Microsoft hololens documentation: Gaze and commit. `https://docs.microsoft.com/en-us/windows/mixed-reality/design/gaze-and-commit`, 2019. Accessed: 2021-05-01.

[24] Microsoft. Sensor data access. `https://docs.microsoft.com/en-us/windows/mixed-reality/develop/platform-capabilities-and-apis/research-mode`, 2019. Accessed: 2021-05-01.

[25] Microsoft. Head-gaze and eye-gaze input in directx. `https://docs.microsoft.com/en-us/windows/mixed-reality/develop/native/gaze-in-directx`, 2021. Accessed: 2021-05-01.

[26] Microsoft. What is mixed reality. `https://docs.microsoft.com/en-us/windows/mixed-reality/discover/mixed-reality`, 2021. Accessed: 2021-05-01.

[27] E. Miluzzo, A. Varshavsky, S. Balakrishnan, and R. R. Choudhury. Tapprints: your finger taps have fingerprints. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pp. 323–336. ACM, 2012.

[28] Y. S. Ryu, D. H. Koh, B. L. Aday, X. A. Gutierrez, and J. D. Platt. Usability evaluation of randomized keypad. *Journal of Usability Studies*, 5(2):65–75, 2010.

[29] F. Schaub, R. Deyhle, and M. Weber. Password entry usability and shoulder surfing susceptibility on different smartphone platforms. In *Proceedings of the 11th international conference on mobile and ubiquitous multimedia*, pp. 1–10, 2012.

[30] SecLists. 100 most common passwords. `https://github.com/danielmiessler/SecLists/blob/master/Passwords/Common-Credentials/10-million-password-list-top-100.txt`, 2020. Accessed: 2021-05-01.

[31] P. Seeburger. Intersection of a line and a plane. `https://math.libretexts.org/Bookshelves/Calculus/Supplemental_Modules_(Calculus)/Multivariable_Calculus`, 2020. Accessed: 2021-05-01.

[32] F. Song, I. Markwood, Y. Liu, S. Zhao, Z. Lu, and H. Zhu. No training hurdles: Fast training-agnostic attacks to infer your typing. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1747–1760. ACM, 2018.

[33] J. Sun, X. Jin, Y. Chen, J. Zhang, Y. Zhang, and R. Zhang. Visible: Video-assisted keystroke inference from tablet backside motion. In *NDSS*, 2016.

[34] F. Tari, A. A. Ozok, and S. H. Holden. A comparison of perceived and real shoulder-surfing risks between alphanumeric and graphical passwords. In *Proceedings of the second symposium on Usable privacy and security*, pp. 56–66, 2006.

[35] P. Wang, S. Zhang, X. Bai, M. Billinghurst, W. He, S. Wang, X. Zhang, J. Du, and Y. Chen. Head pointer or eye gaze: Which helps more in mr remote collaboration? In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 1219–1220, 2019. doi: 10.1109/VR.2019.8798024

[36] Z. Xu, K. Bai, and S. Zhu. Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors. In *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, pp. 113–124. ACM, 2012.

[37] L. Yang and Z. Li. aleak: Privacy leakage through context-free wearable side-channel. In *IEEE Conference on Computer Communications*,

pp. 1232–1240. IEEE INFOCOM, 2018.

[38] G. Ye, Z. Tang, D. Fang, X. Chen, K. I. Kim, B. Taylor, and Z. Wang. Cracking android pattern lock in five attempts. In *Proceedings of the 2017 Network and Distributed System Security Symposium 2017 (NDSS 17)*. Internet Society, 2017.

[39] Z. Zhang, M. Sun, B. Gao, and L. Wang. 2-thumbs typing: A novel bimanual text entry method in virtual reality environments. In *2021 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pp. 530–531, 2021. doi: 10.1109/VRW52623.2021.00147

[40] W. Zhou, Y. Zhou, X. Jiang, and P. Ning. Detecting repackaged smartphone applications in third-party android marketplaces. In *Proceedings of the second ACM conference on Data and Application Security and Privacy*, pp. 317–326, 2012.

[41] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang. Hey, you, get off of my market: detecting malicious apps in official and alternative android markets. In *NDSS*, vol. 25, pp. 50–52, 2012.